Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 04-037

Scalable Partitioning Algorithms for FPGAs with Heterogeneous Resources

Navaratnasothie Selvakkumaran, Abhishek Ranjan, Salil Raje, and George Karypis

September 29, 2004

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
29 SEP 2004		**-**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Scalable Partitioning Algorithms for FPGAs with Heterogeneous Resources	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Army Research Laboratory,2800 Powder Mill Road,Adelphi,MD,20783-1197**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT
see report

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**25**	

Scalable Partitioning Algorithms for FPGAs with Heterogeneous Resources *

Navaratnasothie Selvakkumaran[+], Abhishek Ranjan[++], Salil Raje[++], and George Karypis[+]

[+]Department of Computer Science and Engineering, University of Minnesota
{selva, karypis}@cs.umn.edu
[++]HierDesign Inc, Santa Clara, CA
{ranjan, salil}@hierdesign.com

Abstract

As FPGA densities increase, partitioning-based FPGA placement approaches are becoming increasingly important as they can be used to provide high-quality and computationally scalable placement solutions. However, modern FPGA architectures incorporate heterogeneous resources, which place additional requirements on the partitioning algorithms because they now need to not only minimize the cut and balance the partitions, but also they must ensure that none of the resources in each partition is over-subscribed. In this paper, we present a number of multilevel multi-resource hypergraph partitioning algorithms that are guaranteed to produce solutions that balance the utilization of the different resources across the partitions. We evaluate our algorithms on twelve industrial benchmarks ranging in size from 5,236 to 140,118 cells and show that they achieve minimal degradation in the min-cut while balancing the various resources. Comparing the quality of the solution produced by some of our algorithms against that produced by hMeTiS, we show that our algorithms are capable of balancing the different resources while incurring only a 3.3%–5.7% higher cut.

Keywords: Partitioning, Placement, FPGA, Heterogeneous, Multi-constraint

1 Introduction

The partitioning-driven placement framework enables a divide-and-conquer strategy by successively bisecting the hypergraph and assigning the partitions to successively geometrically split chip surfaces. The recent development of

*This work was supported in part by NSF CCR-9972519, EIA-9986042, ACI-9982274, ACI-0133464, and ACI-0312828; the Digital Technology Center at the University of Minnesota; and by the Army High Performance Computing Research Center (AHPCRC) under the auspices of the Department of the Army, Army Research Laboratory (ARL) under Cooperative Agreement number DAAD19-01-2-0014. The content of which does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred. Access to research and computing facilities was provided by the Digital Technology Center and the Minnesota Supercomputing Institute.

high-quality multilevel hypergraph partitioning algorithms [12, 2] has rekindled interest in this placement methodology and has led to the development of highly scalable and high-quality ASIC placement tools such as Capo [4], Dragon [27], FengShui [19], and TheTo [22]. In contrast, in the context of FPGA-based designs, partitioning-driven placement has not been a popular option due to their historically lower densities and the fact that simulated-annealing based placement tools produce high-quality solutions (e.g., VPR [3]), and can be easily modified to obey complex constraints [10]. However, increasing FPGA densities and the resulting scalability challenges force FPGA CAD tool developers to increasingly rely on more scalable techniques that are now commonly used in the ASIC domain [24].

Adopting a partitioning-driven placement framework to modern FPGAs is not straightforward. Unlike ASIC placement, where the various modules can be placed anywhere, with the only partitioning constraint of balancing the area of the cells assigned to different partitions, modern FPGA architectures incorporate heterogeneous resources such as CLBs, FFs, Multipliers, RAM blocks, IP Cores [29], *etc*. This places additional constraints on the types of partitionings that need to be computed, as the partitioning algorithm must now ensure that the resources used in each partition can be accommodated by the resources available at the different regions of the FPGA. For example, a partitioning solution that places most of the FFs on the one side of the bisection and most of the RAM blocks on the other side of the bisection, even if it is balanced in terms of the total number/area of cells on either side of the cut, is not very useful for FPGA placement as it may over-subscribe these two resource types. To illustrate this, we used a multilevel hypergraph partitioning algorithm (hMeTiS [15]) to bisect twelve different circuits synthesized for the Xilinx Vertex II architecture containing cells that map to different resources. These bisections were computed by ignoring the resource type of each individual cell and using a 2% size difference constraint (*i.e.*, a 49%-51% cut). The resulting bisections, whose characteristics are summarized in Table 1, showed that even though the number of cells assigned to each partition achieves the desired balance constraint, the individual resources are considerably more unbalanced. In most circuits there are resource types whose size difference is over 10% and in eight of the twelve circuits, well over 50% of the different resource types violate the 2% size difference constraint.

Even though the inability of existing hypergraph partitioning models to capture the multi-resource requirements of emerging FPGA architectures has been known for a while [21], there has been relatively little work on developing multi-resource aware partitioning algorithms. To our knowledge, the only exception is the work by Liu, Zhu, and Wong [21] in which they attempted to solve the heterogeneous resource partitioning problem using an FM-based framework. They use an incremental network flow computation based feasibility checking algorithm integrated with a flat FM based graph partitioning algorithm. This approach was able to achieve the resulting resource balancing requirements. However, since the underlying algorithm relies on network flow computations to calculate and update gain values for the FM iteration, its time complexity is $O(n^2)$. This limits the applicability of this algorithm to only very small partitioning instances. In addition to this FPGA-focused work, a number of researchers have looked at the somewhat related problem (see the discussion in Section 7) of partitioning for heterogeneous multi-chip systems [18, 20]. These approaches rely on relatively expensive optimization methods based on functional replication [18] and genetic algorithms [20] and formulate the problem as a complex optimization problem. However, these approaches cannot be directly applied to our context and the high computational complexity of the underlying optimization methods make them impractical for large designs.

	number of resource types	minimum size difference	maximum size difference	average size difference	number of violated resources
ind1	11	0.4%	10.3%	4.4%	6
ind2	9	0.6%	9.5%	4.8%	6
ind3	11	0.9%	27.1%	6.4%	7
ind4	12	0.8%	81.5%	10.6%	9
ind5	11	0.8%	16.6%	5.8%	7
ind6	11	0.5%	13.8%	4.3%	5
ind7	11	0.7%	11.0%	3.0%	3
ind8	12	0.7%	7.6%	2.6%	4
ind9	11	0.9%	33.2%	5.3%	6
ind10	5	0.8%	3.1%	1.6%	1
ind11	11	0.8%	11.1%	3.3%	4
ind12	11	1.2%	30.9%	5.6%	8

Table 1: The characteristics of the bisections produced by hMETiS using an overall 2% size difference constraint. Each circuit contains cells of different type. The number of cell-types is shown in column 2. The extent to which each bisection can balance the individual resource types was measured by computing the size difference for each resource type. The minimum, maximum, and average size difference over the different resource types for each circuit are shown in columns 3–5. Size differences that are less than 2% are considered to satisfy the size difference constraint, whereas differences that are greater than 2% are considered to violate the constraint. The last column shows the number of resource types in each circuit that violate the 2% size difference constraint.

In this paper, we present a new class of scalable *multi-resource hypergraph bisectioning algorithms* that are capable of producing a partitioning solution that simultaneously balances the different resources assigned to each one of the partitions and can be used to implement partitioning-based placement methodologies for emerging FPGA architectures. Specifically, we present five different multi-resource partitioning algorithms that are based on the multilevel hypergraph partitioning paradigm. Three of these algorithms solve the problem by balancing the different resources at the same time as they compute the bisection, while the other two are used to post-process a high-quality but potentially unbalanced solution to enforce the multiple balancing constraints. We experimentally evaluated the performance of these algorithms on twelve different industrial circuits containing up to 140,118 cells. Our results show that each one of these algorithms is capable of producing solutions that satisfy the multiple balancing constraints and achieve different time-quality trade-offs. Moreover, comparing the quality of the solution produced by some of our algorithms against that produced by hMETiS, we show that our algorithms are capable of balancing the different resources while incurring only a 3.3%–5.7% higher cut and requiring only two to five times more time than that required by the hMETiS algorithm. A shorter version of this paper was presented DAC 2004 [23].

The rest of this paper is organized as follows. Section 2 defines various concepts and terms that are used in the paper and presents a brief overview of the multilevel partitioning paradigm. Section 3 provides a formal definition of the multi-resource partitioning problem. Sections 4 and 5 describe the various native and enforcement-based multi-resource partitioning algorithms that we developed. Section 6 presents a comprehensive experimental evaluation of these algorithms. Finally, Section 7 provides some concluding remarks and discusses additional applications of the algorithms presented in this paper.

2 Notation and Background

A *hypergraph* $G = (V, E)$ is a set of vertices V and a set of hyperedges E. Each hyperedge is a subset of the set of vertices V. The *size* of a hyperedge is the cardinality of this subset. A vertex v is said to be *incident* on a hyperedge e, if $v \in e$. Each vertex v and hyperedge e has a weight associated with them and they are denoted by $w(v)$ and $w(e)$, respectively. A circuit/netlist consisting of a set of cells and a set of nets can be directly represented via a hypergraph, whose vertices corresponds to the cells and whose hyperedges corresponds to the nets. Due to this one-to-one correspondence between hypergraphs and netlists we will use the terms vertices/cells and hyperedges/nets interchangeably throughout this paper.

A bisection of V is denoted by a vector P such that $P[i]$ indicates the partition number that vertex i belongs to. The *cut* of the bisection is equal to the sum of the weight of the hyperedges that connect vertices belonging to different partitions. We say that a bisection P of V *satisfies* a *single balancing constraint* specified by $[l, u]$, where $l < u$, iff $l \leq \sum_{v \in V_i} w(v) \leq u$, for each partition V_i. A bisection that satisfies the constraint is called *feasible*, otherwise it is *infeasible*. Given these definitions, the hypergraph bisection problem is formally defined as follows: *Given a hypergraph* $G = (V, E)$ *and a balancing constraint* $[l, u]$, *find a feasible bisection* P *of* G *that minimizes the cut.* Since there is only a single balancing requirement, this formulation is usually referred to as the single-constraint bisectioning problem [8].

Even though a variety of algorithms have been developed for solving the hypergraph bisectioning problem, the current state-of-the-art algorithms follow the multilevel partitioning paradigm [12, 2, 13, 7], as they provide high-quality solutions, have low computational requirements, and can scale to very large hypergraphs. Multilevel partitioning algorithms compute a partitioning of a hypergraph in three phases, commonly referred to as the *coarsening*, *initial partitioning*, and *uncoarsening and refinement* phases. During the coarsening phase, they obtain a sequence of successive approximations of the original hypergraph. Each one of these approximations represents a problem whose size is smaller than the size of the original hypergraph. This process continues until a level of approximation is reached in which the hypergraph contains only a few tens of vertices. At this point, these algorithms enter the initial partitioning phase, which computes a partitioning of that hypergraph. Since the size of this hypergraph is quite small, even simple algorithms such as Kernighan-Lin (KL) [17] or Fiduccia-Mattheyses (FM) [9] lead to reasonably good solutions. Finally, during the uncoarsening and refinement phase, these algorithms take the partitioning computed at the smallest hypergraph and use it to derive a partitioning of the original hypergraph. This is usually done by propagating the solution through the successive better approximations of the hypergraph and using simple approaches to further refine the solution.

3 Problem Definition

Historically, FPGA devices contained resources of single type (*e.g.*, CLBs) that were uniformly distributed throughout the chip. However, taking advantage of ever-increasing silicon densities, modern FPGA devices are fabricated with multiple types of resources, which allow them to efficiently implement complex and high performance designs. One

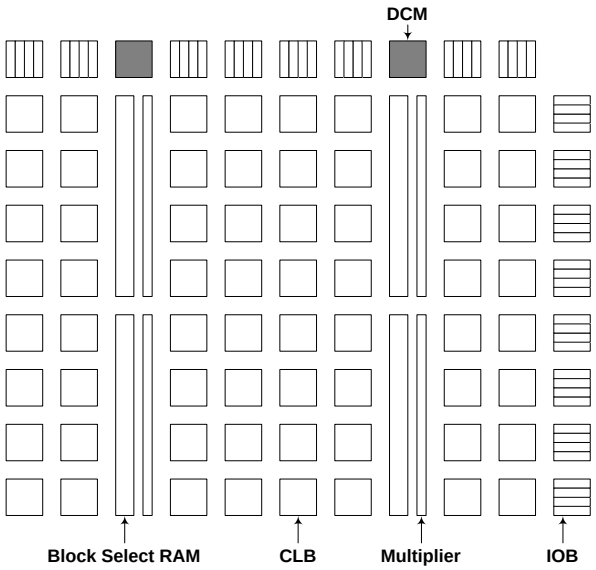

Figure 1: A Section of Virtex II (Xilinx ©) architecture showing heterogeneous resources.

such example is the recently introduced Virtex II architecture from Xilinx (Figure 1) that contains specialized resources such as multipliers and RAM blocks interspersed among CLBs. Similar heterogeneity can be seen in devices such as Altera's Excalibur and Stratix. As a result, design flows created for such modern FPGAs try to pro-actively make use of these specialized resources in order to obtain better performance and versatility.

For partitioning driven placement to succeed in utilizing these different resource types, the partitioning algorithms need to take them into account and compute a solution that minimizes the cut while balancing each type of cells across the cut lines. For example, in the case of the multi-resource hypergraph shown in Figure 2, the bisection denoted by the dashed lines corresponds to a valid solution as it balances each of the resource types individually, whereas the bisection denoted by the doted lines is not valid as it assigns all the cells of type t_3 to one of the two partitions.

Motivated by this observation we focus on multi-resource aware partitioning, which can be formally defined as follows. Consider an FPGA architecture with m distinct resource types and let $t_1, t_2, \ldots, t_m$ denote the types of cells that need to be matched to the resources labeled as $r_1, r_2, \ldots, r_m$, respectively. Let ${cl_{t_i}}^j$ denote the minimum number of cells of type t_i allowed in partition j, and ${cu_{t_i}}^j$ be the maximum number of cells of type t_i allowed in partition j. Then the multi-resource bisection P of G seeks to minimize the cut subject to:

$$ {cl_{t_i}}^j \leq \sum_{\forall v \in V : P[v]=j \ \ and \ \ t(v)=t_i} 1 \leq {cu_{t_i}}^j, \tag{1} $$

where $j = 1, 2$, $i = 1, 2, \ldots, m$, and $t(v)$ is the type of cell v. The partitioning that satisfies Equation 1 is referred to as *feasible* (or *legal*) bisection. Note that this is a general definition of the multi-resource bisection and only the upper bound is usually needed in most cases. Furthermore, when the number of cells of a certain type are small and an odd number, it is sometimes impossible to satisfy the balance constraint. In such cases the balance constraint needs to be

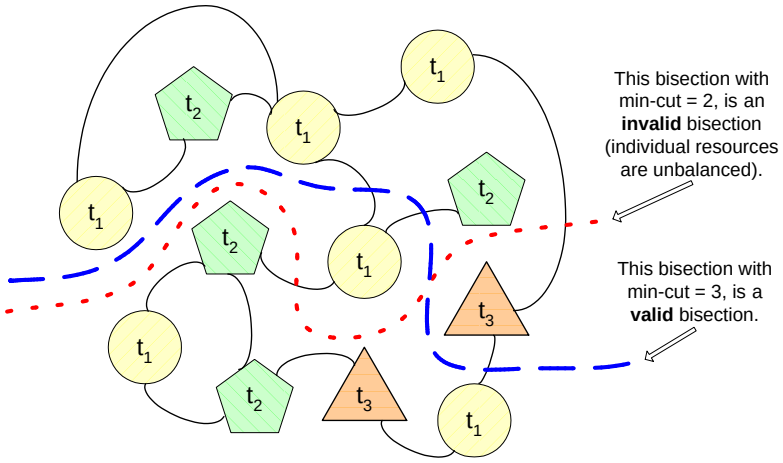

Figure 2: An example multi-resource graph partitioning problem.

relaxed. For example, if there are only 3 cells of a certain type present, then the balance constraint of 49% - 51% is impossible to satisfy, and needs to be relaxed to 33% - 67% for this type of cells, to accommodate them.

Note that the space of feasible solutions for the multi-resource partitioning problem is in general a subset of the feasible solution space when each cell is treated to be part of the same resource type. As a result, the quality of the feasible multi-resource bisections will tend to be worse than that of the feasible single-resource bisections. This is illustrated in Figure 2 in which the feasible solution with respect to the three resources has a cut of three, whereas the feasible solution that ignores the individual cell types has a cut of two.

To solve the multi-resource bisectioning problem we developed two classes of multi-resource partitioning algorithms. The first class, computes the overall solution by constructing a bisection that simultaneously balances the multiple resources, whereas the second class, achieves the desired balance by modifying a bisection that was initially obtained using a traditional single-constraint bisectioning algorithm. We will refer to the first class as the *native multi-resource partitioning* algorithms and to the second class as the *multi-resource enforcement* algorithms. The details of the various algorithms in each of these classes are provided in the following two sections.

4 Native Multi-Resource Partitioning Algorithms

We developed three different algorithms, called *multi-phase*, *multi-constraint*, and *multi-phase–multi-constraint* that are capable of directly computing a partitioning that balances the different resources. These algorithms are motivated by the recently developed graph partitioning algorithms for partitioning finite element meshes arising in multi-phase and multi-physics scientific numerical simulations [16, 25]. Specifically, our *multi-constraint* algorithm is based on the graph partitioning algorithm proposed in [16], our *multi-phase* algorithm is based on the graph partitioning algorithm proposed in [25], whereas the *multi-phase–multi-constraint* algorithm combines elements from both of these approaches. Details on these algorithms are provided in the remainder of this section.

4.1 Multi-Constraint Bisection (MC)

The multi-constraint partitioning formulation, initially developed in the context of graphs [16], was introduced to model the requirements of applications that need to partition graphs that balance multiple quantities associated with their vertices. Formally, the multi-constraint *hypergraph* bisectioning problem formulation in the case of m constraints is defined as follows. Let $H = (V, E)$ be a hypergraph such that each vertex v has a weight vector $\boldsymbol{w}(v)$ of length m associated with it. The ith component of this vector ($w_i(v)$) corresponds to the weight associated with the ith constraint. Without loss of generality, we assume that the weight vectors of the vertices satisfy the property that[1] $\sum_{\forall v \in V} w_i(v) = 1.0$ for $i = 1, 2, \ldots, m$. Using a framework analogous to that used for single-constraint problems (Section 2), let $[l_i, u_i]$ for $i = 1, 2, \ldots, m$, be m lower- and upper-bound constraints on the size of each partition such that $0 \leq l_i \leq u_i$ and $l_i + u_i = 1$. Given these definitions, the goal of the multi-constraint hypergraph bisection problem is to compute a bisection P of V that minimizes the sum of the weight of the hyperedges that span multiple partitions subject to the constraint that

$$l_i \leq \sum_{\forall v \in V : P[v] = j} w_i(v) \leq u_i, \;\; j = 1, 2, \text{ and } i = 1, 2, \ldots, m.$$

As an example of where this problem formulation can be used, consider an application that needs to compute a bisection of a circuit such that the area, power consumption, and pin-count are simultaneously balanced across the two partitions. This partitioning requirement can be formulated as a multi-constraint problem by assigning to each cell a vector of three weights corresponding to the cell's area, power consumption, and pin-count, respectively, and setting $[l_i, u_i] = [0.50, 0.50]$ for $i = 1, 2, 3$. The resulting multi-constraint bisection will produce a bisection that minimizes the cut and also balances each of these three quantities across the two partitions.

However, the multi-constraint formulation can also be used to solve the multi-resource partitioning problem by creating a constraint for each different type of cells. Specifically, given a multi-resource hypergraph $H = (V, E)$ with m different types of cells, each vertex $v \in V$ is assigned a vector of m weights $\boldsymbol{w}(v)$, such that $w_{t(v)}[v] = 1$ and $\forall i \neq t(v), w_i(v) = 0$. For example, in a hypergraph with three types of cells t_1, t_2, and t_3, we would assign $(1, 0, 0)$ to the cells of type t_1, $(0, 1, 0)$ to the cells of type t_2 and $(0, 0, 1)$ to the cells of type t_3. It is easy to see that a feasible multi-constraint partitioning of this problem will correspond to a feasible solution for the multi-resource partitioning problem, as well. This representation and the resulting feasible partitions that it produces are illustrated in Figure 3.

Motivated by the above observation, we developed a multilevel multi-constraint partitioning algorithm for hypergraphs and used it to solve the multi-resource partitioning problem. Specifically, we developed algorithms for the coarsening, initial partitioning, and uncoarsening phases that combine elements of the single-constraint hypergraph partitioning algorithms in hMETiS with the multi-constraint extensions introduced for graph partitioning [16].

[1] In cases in which this does not hold, the weight of each vertex with respect to each constraint can be divided by the total weight of this constraint over all the vertices. The resulting weights will satisfy this condition and the scaling does not affect the ability to find partitioning solutions that satisfy the desired balancing constraints.

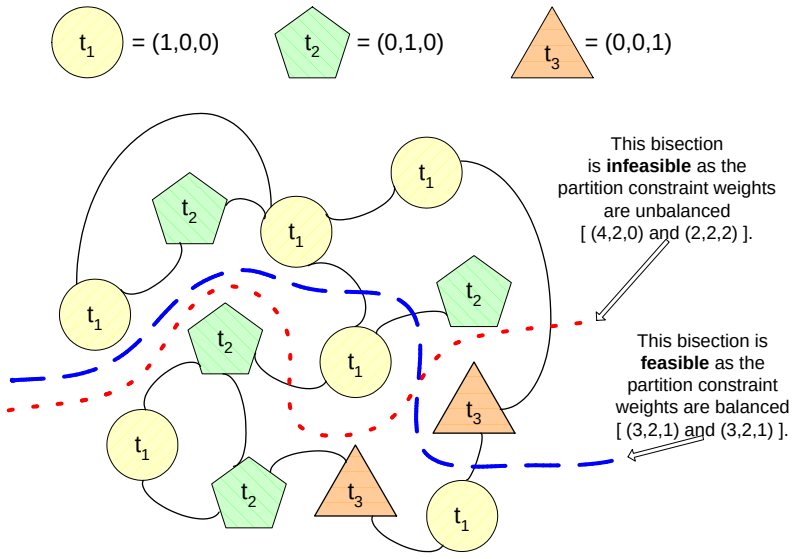

Figure 3: The example multi-resource problem posed as multi-constraint problem.

4.1.1 Coarsening Phase

During the coarsening phase, a sequence of successively smaller hypergraphs is constructed by finding groups of vertices and merging them together to form the vertices of the next level coarser hypergraph. A number of schemes have been developed for selecting what groups of vertices will be merged together to form single vertices in the next level coarse hypergraphs [13, 12, 2, 28]. Of these schemes, the ***first-choice*** (FC) scheme [13], has been experimentally shown to produce high-quality bisections, and forms the basis of the coarsening scheme used in our algorithm.

The FC scheme is derived by modifying the commonly used edge-coarsening (EC) scheme in which a vertex is randomly selected and it is merged with a highly connected and unmatched neighbor. The connectivity to the neighbors is estimated by representing each hyperedge by a clique of edges each with a weight of $w(e)/(|e|-1)$ and by summing the weights of the edges common to each neighbor and the vertex in consideration. However, the FC scheme differs from EC in that it relaxes the requirement that a vertex is matched only with another unmatched vertex. Specifically, in the FC coarsening scheme, the vertices are again visited in a random order. However, for each vertex v, all vertices (both matched and unmatched) that belong to hyperedges incident to v are considered and the one that is connected via the edge with the largest weight is matched with v, breaking ties in favor of unmatched vertices.

The primary purpose of the coarsening phase in single-constraint multilevel partitioning algorithms is to create successively smaller hypergraphs that contain a progressively smaller number of hyperedges from the original hypergraph. In fact, compared to the other coarsening schemes, the FC scheme tends to remove a larger amount of the *exposed hyperedge-weight*; thus making it easier to find high-quality initial partitionings that require little refinement during the uncoarsening phase. However, within the context of the multi-constraint partitioning problem, one can also use the coarsening process to try to reduce the inherent difficulty of the load balancing problem resulting from the presence of multiple weights. This is because it is easier to compute a balanced partitioning if the values of the

different elements of every weight vector are not significantly different from each other, as in this case the balancing problem becomes closer to that of a single-constraint partitioning. Motivated by this observation, we developed a new coarsening scheme, referred to as *balanced first-choice* (BFC), that inherits the key properties and structure of the FC scheme but collapses together sets of vertices that are both well-connected and also lead to more balanced weight vectors.

Specifically, the BFC scheme considers the various vertices in a random order. For each vertex v, it identifies the vertex u that is connected to v with the highest weight edge (let $w(v, u)$ be the weight of this edge). Let $N(v, w(v, u))$ be the set of vertices adjacent to v such that $x \in N(v, w(v, u))$ iff $w(v, x) \geq \alpha w(v, u)$ for $0 \leq \alpha \leq 1$. That is, the set $N(v, w(v, u))$ contains all the vertices that are connected to v via an edge whose weight is greater than a certain fraction of the highest-weight incident edge of v. Each of the vertices in $N(v, w(v, u))$ represents a potential vertex which v can be matched with and among them the BFC scheme selects the vertex x that leads to the most balanced merged vertex. The *balance* of a weight vector is measured as the ratio of the maximum weight over the average weight of the m vector elements. Within this algorithm, the value of parameter α controls the extent to which we emphasize high connectivity over good balance. If α is set close to 1.0, then $N(v, w(v, u))$ will tend to contain few vertices, and thus the algorithm will focus more toward selecting well-connected vertices; whereas if α is close to 0.0, then $N(v, w(v, u))$ will tend to contain most of v's adjacent vertices, emphasizing balance. Our experimentation with this parameter showed that the BFC scheme achieves consistently better results when α is in the range of $[0.85, 0.95]$, as it provides a good balance between reducing the exposed hyperedge weight and also making it easier to find a balanced partitioning. For this reason, in all of our experiments we use $\alpha = 0.90$.

4.1.2 Initial Partitioning Phase

The goal of the initial partitioning phase of a multilevel algorithm is to compute a feasible bisection of the coarsest hypergraph that minimizes the cut. In the case of single-constraint partitioning, ensuring the feasibility of the resulting bisection (i.e., that it satisfies the balancing constraint) is quite straightforward. However, in the context of multi-constraint partitioning, finding a feasible bisection that balances the multiple weights is considerably much harder.

Our algorithm for computing a bisection in the presence of multiple weights is similar in spirit to the greedy region growing algorithm [14] used for computing a bisection of a graph when there is a single weight. In the case of single-weight graphs, the greedy region growing algorithm works as follows. Let $H = (V, E)$ be the hypergraph that we want to bisect into two subgraphs $H_A = (V_A, E_A)$ and $H_B = (V_B, E_B)$. This algorithm initially selects a vertex $v \in V$ randomly, and sets $V_A = \{v\}$ and $V_B = V/V_A$, and then inserts all the vertices $u \in V_B$ into a max-priority queue according to their *gain* function. The gain of a vertex v is the reduction in the value of the hyperedge-cut achieved by moving v from the partition that it belongs to the other partition. Then, it repeatedly selects the top vertex u from the priority queue, moves it to H_A, and updates the priority queue to reflect the new gains of the vertices adjacent to u. The algorithm terminates as soon as the weight of the vertices in H_A becomes more than half of the weight of the vertices in H.

In the case of hypergraphs with multiple vertex-weights, we modified the above algorithm as follows. Instead of using a single priority queue we use m separate queues, where m is the number of weights. A vertex belongs to only

a single priority queue depending on the relative order of the weights in its weight vector. In particular, a vertex v with weight vector $(w_1(v), w_2(v), \ldots, w_m(v))$ belongs to the jth queue if $w_j(v) = \max_i(w_i(v))$. The existence of these multiple priority queues also changes how the vertices are selected and moved from H_B to H_A. At any given time, depending on the relative order of the weights of hypergraph H_B, the algorithm moves the vertex from the top of a specific priority queue. In particular, if $w_j(V_B) = \max_i(w_i(V_B))$, then the jth queue is selected. If this queue is empty, then the non-empty queue corresponding to the next heavier weight is selected, and so on. The algorithm terminates as soon as one of the weights of H_A become more than the user specified lower-bound for the corresponding weight of H.

A key question that needs to be addressed is the extent to which the above algorithm can lead to a feasible solution or not. The problem of computing a balanced partitioning of a graph in the presence of multiple vertex weights has been analyzed in [16], in which they developed a multi-constraint bin-packing algorithm capable of splitting a set of n multi-weight objects into two bins. The authors showed that, under reasonable assumptions, their multi-constraint bin-packing algorithm can lead to partitions whose relative imbalance is bounded by $mw_{\max}$, where m is the number of weights, and $w_{\max}$ is the largest individual weight among the m weights across the n objects. When $m = 2$, this multi-constraint bin-packing algorithm uses the same pair of priority queues as our initial partitioning algorithm and also follows similar rules for selecting the queue that will provide the next vertex for inclusion in H_A. As a result, our initial partitioning algorithm can achieve very good bounds on the relative balance of the two partitions and, as such, it leads to a feasible solution. In addition, by preferring the highest gain vertices from the selected queue, our algorithm extends the multi-constraint bin-packing algorithm to compute a bisection, which besides being feasible, also tries to minimize the hyperedge-cut in a greedy fashion.

However, for $m > 2$, a precise implementation of the weight selection algorithm used in the multi-constraint bin-packing algorithm is quite complex, as it requires maintaining $m!$ priority queues. For this reason, our initial bisection algorithm is only a loose approximation of the multi-constraint bin-packing algorithm of [16]. Consequently, it does not provide any useful bounds on the relative balance of the two partitions, and in some cases it may fail to achieve reasonably tight balance. To address this problem, after we compute the bisection, we perform an explicit balancing step followed by a bisection refinement step to obtain both better balance and also further improve the quality of the bisection. The algorithms used to balance and refine a multi-constraint bisection are described in Section 4.1.3. We repeat this process a small number of times and pick the best solution as our initial solution.

4.1.3 Uncoarsening Phase

During the uncoarsening phase, a partitioning of the coarser hypergraph is successively projected to the next level finer hypergraph, and a partitioning refinement algorithm is used to optimize the objective function without violating the balancing constraints. A class of local refinement algorithms that tend to produce very good results when the vertices have a single weight [12], are those that are based on the FM algorithm [9]. The FM algorithm starts by inserting all the vertices into two max-priority queues, one for each partition, according to their gains. Initially all vertices are *unlocked*, *i.e.*, they are free to move to the other partition. The algorithm iteratively selects an unlocked vertex v from the top of the priority queue from one of the partitions (source partition) and moves it to the other partition (target

partition). The source partition is determined based on whether the current bisection is a feasible solution or not. If it is feasible, then the partition that contains the highest gain vertex becomes the source. On the other hand, if it is not feasible (*i.e.*, the balancing constraint is violated), the partition that contains the largest number of vertices, becomes the source. When a vertex v is moved, it is *locked* and the gain of the vertices adjacent to v are updated. After each vertex movement, the algorithm records the value of the objective function achieved at this point and whether or not the current bisection is feasible or not. A single pass of the algorithm ends when there are no more unlocked vertices. Then the recorded values of the objective function are checked, and the point where the minimum value was achieved while achieving a feasible solution is selected, and all vertices that were moved after that point are moved back to their original partition. This now becomes the initial partitioning for the next pass of the algorithm.

Multi-constraint FM Refinement (MC-FM) For the purpose of refining a bisection in the presence of multiple balancing constraints, we have developed a new bisection refinement algorithm, called MC-FM, that uses FM's overall strategy and operates as follows. For each one of the two partitions, we maintain m priority queues, where m is the number of weights. A vertex belongs to only a single priority queue depending on the relative order of the weights in its weight vector. In particular, a vertex v with weight vector $(w_1(v), w_2(v), \ldots, w_m(v))$, belongs to the jth queue if $w_j(v) = \max_i(w_i(v))$. Given these $2m$ queues, the algorithm starts by initially inserting all the vertices to the appropriate queues according to their gains. Then, it proceeds by selecting one of these $2m$ queues, picking the highest gain vertex from this queue, and moving it to the other partition. The queue is selected as follows. If the current bisection represents a feasible solution, then the queue that contains the highest gain vertex among the $2m$ vertices at the top of the priority queues is selected. On the other hand, if the current bisection is infeasible, then the queue is selected depending on the relative weights of the two partitions. Specifically, if A and B are the two partitions, then the algorithm selects the queue corresponding to the largest $w_i(x)$ with $x \in \{A, B\}$ and $i = 1, 2, \ldots, m$. If it happens that the selected queue is empty, then the algorithm selects a vertex from the non-empty queue corresponding to the next heaviest weight *of the same partition*.

For example, if $m = 3$, $(w_1(A), w_2(A), w_3(A)) = (.43, .60, .52)$, and $(w_1(B), w_2(B), w_3(B)) = (.57, .4, .48)$, the algorithm will select the second queue of partition A. If this queue is empty, it will then try the third queue of A, followed by the first queue of A. Note that we give preference to the third queue of A as opposed to the first queue of B, even though B has more of the first weight than A does of the third. This is because our goal is to reduce the second weight of A. If the second queue of A is non-empty, we will select the highest gain vertex from that queue and move it to B. However, if this queue is empty, we still will like to decrease the second weight of A, and the only way to do that is to move a node from A to B. This is why when our first-choice queue is empty we select the most promising node from the same partition that this first-queue belongs to.

Multi-constraint Balancing (MC-B) As discussed in Section 4.1.2, the region-growing algorithm used to compute the initial bisection of the hypergraph may fail to produce a feasible solution when $m > 2$. For this reason we developed an iterative algorithm that balances the different weights by moving vertices between the two partitions. This *multi-constraint balancing algorithm* operates in a fashion similar to MC-FM, except that it gives priority to

finding a balanced bisection rather than minimizing the cut.

MC-B uses the same set of priority queues as MC-FM, and in each step it moves the vertex from the priority queue corresponding to the most violated balance constraint. The most violated balance constraint is determined by measuring the constraint weight values in each partition and choosing the constraint whose weight is the furthest away from the minimum required amount of constraint weight. Then a move is selected from the priority queue of the other partition, so that the moving of that particular vertex to the most violated side will reduce the violation. If that particular queue is empty, other queues on the same side are searched for vertices that can reduce the violation of most unbalanced constraint. Note that during the operation of this algorithm, all the moves that improve the balance are accepted irrespective of the degradation in the cut. Thus, this balancing step tends to increase the cut, especially when the number of constraints is large.

4.2 Multi-Phase Bisection (MP)

The multi-phase bisection algorithm is based on the intuitive idea of tackling each type of cells one by one. Essentially, it uses the single-constraint partitioner (hMETIS) to partition a single type of cells while the rest of the types are made non-participatory in the partitioning process. Since each type is handled in separate phase, this algorithm is referred to as *multi-phase bisection*. The cells are made non-participatory either by making them fixed (i.e., not allowing them to move between partitions) or by removing them from the hypergraph. We make this choice based on whether the partition information of a cell is available or not. Specifically, if the cells of a particular type have already been bisected then we make them fixed cells, and if the cells of a certain type have not been partitioned then we make them non-participatory by removing them from the partitioning problem.

The algorithm proceeds as follows. Initially, it constructs a series of sub-hypergraphs denoted by $H_1, H_2, \ldots, H_m$ such that H_k contains only the cells of type $t_1, t_2, \ldots, t_k$. That is, sub-hypergraph H_1 contains only the cells of type t_1, H_2 contains only the cells of type t_1 and t_2, H_3 contains only the cells of type t_1, t_2, and t_3, and so on. Note that H_m is nothing more than the original hypergraph H. Because each of these sub-hypergraphs contains a subset of the cells, we will refer to them as *partial hypergraphs*. This sequence of partial hypergraphs is constructed by starting from H_m and repeatedly removing from it all the cells of type $t_m, t_{m-1}, \ldots, t_2$ in that order, leading to the sub-hypergraphs $H_{m-1}, H_{m-2}, \ldots, H_1$, respectively. Next, the algorithm uses hMETIS to bisect one-by-one each of the sub-hypergraphs $H_1, H_2, \ldots, H_m$ to obtain the partitions $P_1, P_2, \ldots, P_m$, respectively. Specifically, partitioning P_1 is obtained by bisecting H_1 using the balance constraint of $[cl_{t_1}, cu_{t_1}]$. Partitioning P_2 is obtained by first fixing the cells of H_2 that are of type t_1 to either partition one or two based on the previously computed partitioning P_1, and then bisecting the cells of type t_2 that remain using a balance constraint of $[cl_{t_2}, cu_{t_2}]$. Since by construction, the cells of type t_1 in H_2 correspond to the cells of H_1, partitioning P_1 contains all information required to assign these cells to either partition one or two; thus, the above procedure will result in partitioning all the cells of H_2. Moreover, the resulting partitioning satisfies the balance constraints of the cell-types that it contains (i.e., it is a feasible solution) because (i) the cells of type t_1 satisfy the constraint $[cl_{t_1}, cu_{t_1}]$, as they remained fixed during partitioning; and (ii) the cells of type t_2 satisfy the constraint $[cl_{t_2}, cu_{t_2}]$, as it was enforced by hMETIS. Partitions $P_3, \ldots, P_m$ are computed by

following a similar procedure that fixes the cells of the previously partitioned cell-types and uses hMETIS to bisect the newly added cell-type subject to its specific balance constraint. By following a similar argument as it was the case with P_2, it can be easily shown that these partitionings will also correspond to feasible solutions and consequently partitioning P_m will be a feasible partitioning to the overall problem.

Cell-Type Ordering Within the context of this multi-phase partitioning algorithm, there are two elements that affect its overall effectiveness. The first is the method used to order the different types of cells and thus the sequence by which the overall partitioning is computed. In our algorithm, the ordering of the various cell-types is determined by the number of cells of each type that is available. In particular, since the overall multi-resource partitioning is computed in an incremental fashion, in which the partitioning of cell-types $t_1, t_2, \ldots, t_{k-1}$ has an influence on how cells of type t_k are partitioned in H_k, we would like the cell-type ordering to be such that it orders first the cell-types whose bisection will most likely account for a large fraction of the overall cut. By doing so, we allow the algorithm to focus on these *costly* cell-types as early as possible. This allows it to compute high-quality partitionings as they are only affected/constrained by a small number of previously partitioned cell-types. Even though the a priori determination of such an ordering is all but impossible, our experimentation showed that the number of cells of each cell-type correlates well with its contribution to the overall cut. That is, the bisection of the more frequent cell-type will account for a larger fraction of the cut when compared to the less frequent. For this reason, our algorithm orders the different cell-types in non-increasing order of their respective frequencies. This ordering ensures that the cells with known partitioning information is always larger than (or equal) the number of cells without partitioning information during the partitioning of any of the sub-hypergraphs.

Modeling the Modified Nets The second element affecting the performance of the multi-phase partitioning algorithm is the method used to model the nets affected by the elimination of the cells that occurs during the construction of the successively smaller sub-hypergraphs $H_{m-1}, H_{m-2}, \ldots, H_1$. To see why we need to differentiate between these *partial* nets and the nets that remain unaffected (referred to as *complete nets*), consider two equal-size nets e_1 and e_2 of H_k such that e_1 contains all of its original cells whereas e_2 was obtained after eliminating some of its cells. Let us assume that there exist two feasible bisections $P_k^{e_1}$ and $P_k^{e_2}$ of H_k such that both of them cut the same set of nets with the only difference being that $P_k^{e_1}$ cuts e_1 but not e_2, whereas $P_k^{e_2}$ cuts e_2 but not e_1. By construction, both of these bisections are equally good as they cut the same number of nets. However, within the context of our incremental multi-resource partitioning algorithm, $P_k^{e_2}$ should be preferred over $P_k^{e_1}$ because the cut status of e_1 (or that of any other complete net) remains unchanged during the incremental partitioning of the successive sub-hypergraphs $H_{k+1}, \ldots, H_m$, whereas the cut status of e_2 can change. In particular, even if e_2 is not being cut by the current bisection, it can still be cut in successive sub-hypergraphs since they contain more of its original cells. Moreover, the likelihood of changing the future status of an uncut partial net increases as the difference between the original and the partial size of the net increases. This is because partial nets that contain a much smaller numbers of their original cells can be cut in multiple ways and/or in multiple successive sub-hypergraphs (if they contain a large number of different cell-types). For this reason, everything else being equal, it is more preferable to cut a partial net as opposed

to a complete net, and among these partial nets it is better to cut a net that has a large number of its original cells removed. In our algorithm we introduce such preferences by decreasing the weight of each partial net, which bias the bisection toward partial nets. Specifically, for each partial net e', we calculate its new weight $w(e')$ as $w(e) * (|e'|/|e|)$, where e and $w(e)$ are its corresponding complete net and its weight, respectively. Note that this re-weighting formula takes into account the number of cells that have been removed, further biasing the bisection toward partial nets that are much smaller than their original nets.

4.3 Multi-Phase Multi-Constraint (MPMC)

The third native multi-resource partitioning algorithm that we developed, referred to as MPMC, extends the multi-phase algorithm presented in Section 4.2 by incorporating some of the elements of the multi-constraint partitioning algorithm described in Section 4.1 and by augmenting the partial hypergraphs to include additional hyperedges, which are designed to better capture the structure of the original hypergraph.

Post-Bisection Refinement One of the characteristics of the multi-phase partitioning algorithm is that once the partitioning of a particular partial hypergraph H_i has been computed, the locations of the cells of type t_i remain fixed and do not change as additional cell-types are being partitioned. Even though this helps in reducing the overall complexity of the algorithm (each cell is partitioned only once), and in ensuring that the resulting bisection represents a feasible solution, it may produce bisections whose overall cut is relatively high. This can be reduced if we allow the movement of previously partitioned cell-types. This is especially true in cases in which the scheme utilized to order the various cell-types (based on their frequency) fails to correctly predict the relative difficulty of partitioning each cell-type and in cases where there is a high degree of interdependency between the partitionings of different cell-types, and the only way to obtain a low cut bisection is to consider them at the same time. MPMC overcomes these types of problems by using the multi-constraint bisection refinement algorithm described in Section 4.1.3 to refine the bisection of each partial hypergraph. This refinement is applied after the bisection of H_i has been computed and is allowed to move between partitions cells of types $t_1, \ldots, t_i$ (i.e., all the cells that are part of H_i)) as long as such moves improve the cut without violating the balance constraints.

Pseudo-Hyperedge Addition Our initial experiments with the multi-phase partitioning algorithm revealed that as the partial hypergraphs contain a progressively smaller number of different cell-types, their topological structure can change dramatically when compared to that of the original hypergraph. For example, it is not uncommon to obtain partial hypergraphs that have many disconnected components, even when the original hypergraph is relatively well-connected (i.e., it has large balanced cuts). This problem becomes more pronounced in cases where there are a large number of different cell types and/or there is no single cell-type that accounts for a large fraction of the total number of cells. The problem with partial hypergraphs that are structurally different from the original hypergraph is that a good bisection of them may not necessarily lead to a good bisection to the overall problem. For example, if a partial hypergraph contains disconnected components, we may be able to identify a balanced bisection that does not cut any hyperedges. However, this zero-cut bisection when viewed within the larger context of the original hypergraph may

actually be worse than another bisection with a higher cut. Unfortunately, the partial hypergraph contains no information to allow the algorithm to select what appears to be a worse bisection. One way of addressing this problem is to rely on the post-bisection multi-constraint refinement iterations discussed in the previous paragraph. However, the heuristic and local nature of such refinement may not always lead to the best possible solution. For this reason, MPMC takes a complementary approach and augments the original representation of the partial hypergraphs by introducing *pseudo hyperedges* that try to retain the connectivity information that otherwise would be lost in the process of removing cells.

Specifically, these pseudo hyperedges are created as follows. When a cell u is removed, the set of cells that u is connected to (neighbors $N(u)$) is analyzed to determine how closely each of them is connected to u. The degree of connectivity is determined using the same scheme used for measuring the connectivity between a pair of cells during the coarsening phase. That is, we represent the hyperedges as cliques of edges with the weight of $w(e)/(|e|-1)$. The sum of weights of such edges connecting cells u and v ($v \in N(u)$) determines the connectivity between u and v. For all the cells in $N(u)$, that have a connectivity to u higher than a certain threshold are considered to be "highly connected", and are allowed to be connected to a newly introduced pseudo hyperedge. The motivation behind these pseudo hyperedges is to bias the partitioning of the partial hypergraphs towards aligning them with the overall bisection that would have been obtained without the removal of cells. However, in order to ensure that these pseudo hyperedges do not overly bias the bisection, they are assigned a much smaller weight than that of the other hyperedges. Empirically, we found that using the connectivity threshold of 10% of the average hyperedge weight and setting the weight of pseudo hyperedges equal to 10% of the average hyperedge weight improved the overall partitioning results for tighter balance constraints (as shown later in Section 6.1). Furthermore, when determining connectivity, previously added pseudo hyperedges are *not* taken into consideration.

4.4 Additional Improvements

After the bisection of the original hypergraph has been computed, it is possible to further improve the cut by applying a multi-constraint V-cycle. This consists of two components, *restricted multi-constraint coarsening* and multi-constraint refinement. The restricted multi-constraint coarsening step differs from regular multi-constraint coarsening [12] by an additional requirement: any two vertices that are collapsed together must belong to the same partition. The information regarding the partitioning is thus preserved during the creation of successive approximate hypergraphs. The second component of the V-cycle is same as the multi-constraint refinement presented in Section 4.1.3.

5 Multi-Resource Enforcement Algorithms

In analyzing the characteristics of the various multi-resource circuits of Table 1, which correspond to large designs synthesized for the Xilinx Virtex II architecture, we discovered that the different types of cells are reasonably well-distributed throughout the underlying hypergraph. This suggests that the bisections produced by single-constraint partitioning algorithms, though not perfectly balanced, they will not be arbitrarily unbalanced either. Moreover, since

these partitionings can be computed using state-of-the-art multilevel schemes, they will have small cuts. Motivated by this observation, we developed two schemes that take as input a min-cut single-constraint partitioning and modify it to enforce the various multi-resource balance constraints.

Single-Constraint Direct-Balancing (SCDB) The first scheme takes advantage of the multi-constraint balancing and bisection refinement algorithms, which were developed within the context of the multi-constraint partitioning algorithm (Section 4.1), and solves the multi-resource partitioning problem in three steps. First, it treats all the cells as being of the same type and computes a bisection using hMETiS. Second, it uses the multi-constraint balancing algorithm (MC-B) to modify this bisection so that the different cell-types satisfy their respective balancing constraints. Third, it improves the quality of the feasible solution by using the multi-constraint bisection refinement algorithm (MC-FM) to further modify this bisection. Compared to the native multi-resource partitioning algorithms (Section 4), the key advantage of SCDB is that it is considerably faster as it essentially requires a small number of additional FM-style iterations to perform the balancing and refinement steps. However, despite its low computational requirements, as our experiments will later show, its overall performance is remarkably good, and in many cases it is comparable or better than the native schemes.

Single-Constraint Multi-Phase Balancing (SCMB) The second scheme incorporates the idea of enforcing the multi-resource constraints within the context of the multi-phase multi-constraint partitioning framework (Section 4.3). Specifically, let P_h be the single-constraint bisection computed by hMETiS, let $t_1, t_2, \ldots, t_m$ be the m cell-types sorted in increasing *unbalanced* order with respect to P_h, and let t_x ($1 \leq x \leq m$) be the first cell-type that violates the balancing constraint. That is, in the bisection produced by hMETiS, the cells of type t_1 are the most balanced, the cells of type t_m are the least balanced, and the cells of types $t_1, \ldots, t_{x-1}$ satisfy the balancing constraints. The SCMB algorithm creates a sequence of partial hypergraphs $H_x, H_{x+1}, \ldots, H_m$ such that H_{x+i} ($1 \leq x + i \leq m$) contains cells of types $t_1, \ldots, t_{x+i}$ and uses P_h to derive the partitioning of the cells in H_x whose type is less than t_x. From that point onwards, SCMB proceeds in a fashion similar to MPMC, computing a bisection of each successively larger partial hypergraph based on the bisection of the previous partial hypergraph. Essentially SCMB inherits the balanced portions of the initial partitioning computed by hMETiS and then it iteratively partitions the unbalanced cell-types using an order determined by how much each cell-type violates its respective balancing constraint.

6 Experimental Evaluation

We experimentally evaluated our multi-resource aware partitioning algorithms on an industrial benchmark suite consisting of twelve large designs synthesized for the Xilinx Virtex II architecture. The types of cells consist of sub CLB elements such as LUTs, FFs, MUXes, control gates and non CLB elements such as RAM Blocks, DCMs, IOBs, Multipliers etc. We chose sub-CLB elements as they provide more types of elements in the benchmarks, which helps in validating the robustness of our algorithms. The details of these benchmarks are listed in Table 2. The column labeled "# types" shows the number of distinct types of cells available on that particular benchmark. The column labeled

				No. of cells of various types		
	# cells	# nets	# types	min	max	avg
ind1	18160	17689	11	1	8138	1651
ind2	5236	4874	9	3	2584	582
ind3	15783	16272	11	14	5889	1435
ind4	58571	60734	12	6	22193	4881
ind5	89697	91925	11	9	45305	8154
ind6	56462	57674	11	3	26759	5133
ind7	119407	121822	11	5	55873	10855
ind8	136539	139147	12	1	73106	11378
ind9	109115	111776	11	4	54377	9920
ind10	72130	49594	5	58	42789	14426
ind11	92778	93184	11	1	46577	8434
ind12	140118	141505	11	4	76887	12738

Table 2: The characteristics of netlists used

		Without V-cycle			With V-cycle		
	hMETiS	MC	MP	MPMC	MC	MP	MPMC
ind1	246	378	987	403	346	426	388
ind2	149	181	349	149	173	144	129
ind3	101	224	908	169	224	908	169
ind4	153	405	4012	446	376	508	336
ind5	717	1133	2188	1053	1058	1221	1039
ind6	809	1649	2615	1038	1649	2548	1038
ind7	1021	1187	4126	1234	1081	957	1151
ind8	400	682	4076	921	568	707	734
ind9	1392	1577	4937	1832	1491	1651	1656
ind10	480	528	719	550	498	505	528
ind11	373	545	1311	582	504	730	570
ind12	409	636	1300	533	576	744	531
ARQ	1.000	1.554	4.406	1.500	1.448	1.882	1.386
Time	1.000	0.577	0.230	2.496	1.760	2.360	5.206

Table 3: Performance of algorithms as an average of 10 runs for a 49%-51% balance constraint.

"min" shows minimum number of cells of any type for that benchmark, and similarly the "max" and "avg" columns provide the distribution details of number of cells in each hypergraph.

To evaluate the quality of the solutions obtained by the various multi-resource partitioning algorithms, we used hMETiS (version 1.5.3 [15]) to obtain single-constraint bisections of the different hypergraphs. These solutions were obtained using hMETiS's default parameters (including V-cycle at the end). Furthermore, to make such quality comparisons easier, we computed the Average Ratio of Quality (ARQ) of each algorithm against that obtained by hMETiS. To ensure the meaningful averaging of these ratios, we first took the $\log_2$-values of these ratios, then calculated their mean μ, and then used 2^{μ} as their average. This geometric mean of ratios ensures that ratios corresponding to comparable degradations or improvements (*i.e.*, ratios that are less than or greater than one) are given equal importance. The ARQ number larger than 1.0 indicates a degradation in quality.

To ensure the statistical significance of our experimental results, for both hMETiS and each one of the five multi-resource partitioning algorithms we report the average results of ten runs.

		Without V-cycle			With V-cycle		
	hMETiS	MC	MP	MPMC	MC	MP	MPMC
ind1	213	261	940	375	243	337	355
ind2	147	152	316	123	141	103	114
ind3	85	126	922	177	110	128	110
ind4	127	217	3910	241	171	184	149
ind5	634	779	2242	943	739	813	883
ind6	822	924	2390	1022	871	841	932
ind7	917	983	4376	1167	873	849	1059
ind8	430	558	3781	711	502	431	425
ind9	1289	1449	4052	1454	1367	1371	1326
ind10	360	429	543	391	399	376	377
ind11	193	271	1053	237	247	240	236
ind12	307	375	1334	440	361	366	413
ARQ	1.000	1.246	4.811	1.383	1.136	1.141	1.165
Time	1.000	0.636	0.255	2.667	1.806	1.863	5.015

Table 4: Performance of algorithms as an average of 10 runs for a 45%-55% balance constraint.

6.1 Comparison of Native Algorithms

Tables 3 and 4 show the results obtained by the various native multi-resource partitioning algorithms (described in Section 4) for the 49%–51% and the 45%–55% balance constraints, respectively. Each of these tables shows the average minimum cuts obtained by the MC, MP, and MPMC multi-resource partitioning algorithms under two different scenarios. In the first scenario, the solutions obtained by these algorithms were kept as they were, whereas in the second scenario, the solutions were further refined by performing a V-cycle refinement step (as discussed in Section 4.4). In addition, the columns labeled "hMETiS" show the average min-cut obtained by hMETiS for either 49%–51% or 45%–55% balance.[2] Finally, the rows labeled "ARQ" provide the average ratio of quality of each algorithm to hMETiS's results (computed using the scheme described in the previous section), and the rows labeled "Time" show the amount of time required by the multi-resource partitioning algorithms relative to that required by hMETiS. Numbers less than one represent run-times that are smaller than that of hMETiS, whereas numbers greater than one represent higher run-times.

Comparing the results in these tables we can see that all schemes produce solutions whose cuts are worse than those produced by hMETiS. This should not be surprising, as hMETiS solves the single-constraint bisectioning problem which, in general, does not solve the multi-resource partitioning problem.

Comparing the solutions produced by the various multi-resource partitioning algorithms we can see that there is a considerable amount of variability in the quality of the final solutions. In particular, in the absence of V-cycle refinement, the quality of the solutions produced by MP are significantly worse than those produced by either MC or MPMC. On average, the 49%–51% cuts produced by MP are 4.4 times worse than those produced by the single-constraint hMETiS, whereas the cuts produced by MC and MPMC are only 55.4% and 50% worse than hMETiS's cuts, respectively. Similar trends can be also observed for the 45%–55% cuts, as well. These results illustrate that the multi-constraint algorithm (MC) and the modifications to the multi-phase partitioning algorithm implemented in the

[2] hMETiS's bisections will not necessarily solve the multi-resource problem, as they do not account for the different cell types.

		Without V-cycle		With V-cycle	
	hMETiS	SCDB	SCMB	SCDB	SCMB
ind1	246	265	251	260	238
ind2	149	161	165	160	162
ind3	101	125	124	125	124
ind4	153	230	251	226	251
ind5	717	1340	868	799	864
ind6	809	880	827	879	827
ind7	1021	998	1056	997	1048
ind8	400	488	411	472	394
ind9	1392	1463	1439	1456	1438
ind10	480	491	488	489	486
ind11	373	414	374	403	213
ind12	409	499	503	494	503
ARQ	1.000	1.184	1.119	1.123	1.057
Time	1.000	1.075	1.845	1.898	2.945

Table 5: Performance of algorithms combined with multi-constraint V-cycle as an average of 10 runs for a 49%-51% balance factor.

MPMC algorithm lead to superior solutions.

Comparing the results without and with V-cycle refinement we see that the overall quality of all three algorithms improves by using V-cycle refinement. However, the overall rate of improvement is different for different schemes. The MP algorithm gains the most, whereas the MPMC algorithm gains the least. We believe that the reason for that is the fact that the solutions of MC and MPMC are already of reasonably high quality, and thus, there is relatively little room for improvement. However, because MP's initial solution is considerably worse, by applying a V-cycle refinement, we can achieve dramatic quality improvements. As a result, the 49%–51% solution for MP now becomes only 88.2% worse than that of hMETiS.

Finally, comparing MC with MPMC we can see that the latter leads to better solutions for a 49%-51% balance constraint, which are 5%–10% better on average than those obtained by MC. However, this quality advantage comes at the expense of higher computational requirements. In general, MPMC requires 2.5 to 5.0 times more time than that required by MC. Note that hMETiS takes more run time than MC and MP because it performs V-cycle refinement at the end, while MC and MC do not.

6.2 Comparison of Enforcement Algorithms

Tables 5 and 6 show the results obtained by the SCDB and SCMB enforcement-based multi-resource partitioning algorithms (described in Section 5) for a 49%–51% and a 45%–55% balance, respectively. Each of these tables shows the average minimum cuts obtained by the two partitioning algorithms without and with V-cycle refinement. In addition, the columns labeled "hMETiS" show the results obtained by hMETiS (which are identical to those shown in Tables 3 and 4), the rows labeled "ARQ" provide the average ratio of quality of each algorithm to hMETiS's results, and the rows labeled "Time" show the amount of time required by the multi-resource partitioning algorithms relative to that required by hMETiS.

Comparing the solutions produced by the two enforcement-based multi-resource partitioning algorithms on these

		Without V-cycle		With V-cycle	
	hMETiS	SCDB	SCMB	SCDB	SCMB
ind1	213	218	213	216	204
ind2	147	149	150	149	150
ind3	85	99	96	98	95
ind4	127	167	159	149	155
ind5	634	675	665	669	652
ind6	822	848	832	846	831
ind7	917	928	922	902	905
ind8	430	479	430	425	427
ind9	1289	1334	1335	1320	1332
ind10	360	368	364	363	364
ind11	193	212	193	211	192
ind12	307	375	327	363	322
ARQ	1.000	1.088	1.046	1.058	1.033
Time	1.000	1.034	1.278	1.945	2.035

Table 6: Performance of algorithms combined with multi-constraint V-cycle as an average of 10 runs for a 45%-55% balance factor.

two sets of problems we can see that, unlike the native algorithms, there is relatively little variation between the performance achieved by them. Specifically, the performance difference between the two schemes is less than 7%, on the average. However, the SCMB algorithm is consistently better than SCDB, leading to better solutions in 31 out of the 48 different experimental data-points. Comparing the results without and with V-cycle refinement we see that as it was the case with the native algorithms, the overall quality of the two algorithms improves, as well. However, those improvements are relatively small, ranging between 2% and 5% on average. Finally, comparing the amount of time required by these algorithms we can see that SCMB is slower than SCDB, but in most cases the difference is small.

6.3 Overall Comparisons

Comparing the performance achieved by the various multi-resource partitioning algorithms we can see that in almost all cases, the enforcement-based algorithms lead to solutions that have a lower cut than those obtained by the native multi-resource partitioning algorithms. For example, the best-performing enforcement-based scheme (SCMB) outperforms the best-performing native scheme in 41 out 48 data-points. Moreover, the cut differences are considerable, and on the average SCMB leads to cuts that are 13%–32% better than that of MPMC. However, this performance advantage is also data-set dependent, and the relative performance of the various schemes can change for different benchmarks.

Finally, comparing the performance achieved by SCMB against that achieved by the single-constraint hMETiS, we can see that the overall increase in the cut resulting by solving the multi-resource partitioning problem is quite small. For example, if we consider SCMB's results with V-cycle refinement we can see that on average the cut increases by only 5.7% and 3.3% for the 49%–51% and the 45%–55% balance constraints, respectively.

7 Conclusions and Discussion

In this paper we presented two classes of multi-resource aware partitioning algorithms for enabling partitioning-based placement methods for FPGA architectures with heterogeneous devices. These algorithms are very effective in minimizing the cut while satisfying multiple balancing requirements with acceptable computational effort. The average cut of the most effective algorithm is only 5.7% and 3.3% worse than that of the state-of-the-art partitioning tool hMETIS [15] for the 49%–51% and the 45%–55% balance constraints, respectively. Moreover, their additional computational requirements are small, requiring only two to three times more time than hMETIS. Moreover, as in most real-life placement applications we are interested in finding partitionings that do not *over-subscribe* each specific resource type, the actual balance constraints for each resource can be set differently. This will increase the space of feasible solutions and will allow the algorithms presented here to find even higher quality partitionings. These results illustrate that high-quality partitionings are feasible for designs with multiple resource requirements, suggesting that partitioning-based placement methods can be used for placing such designs on modern FPGA architectures.

Even though the key motivation behind this research is partitioning-driven placement of FPGA architectures with heterogeneous resources, the algorithms developed can be used to solve a number of other problems encountered in today's complex chip layout. One such potential application occurs in properly handling area-array I/Os. Traditionally, I/O pins are located on the periphery of the chip. However, some of the modern fabrication requirements place I/O pins throughout the core area. This type of layout is named "area-array I/O" layout [5]. Partitioning-driven placement for such layouts require the ability to compute partitionings in which both the I/O pins as well as the core cells are well-distributed through the physical design—a task that can be achieved by the multi-resource partitioning algorithms developed in this paper. Another application arises while computing layouts that can ensure signal integrity. Specifically, one of the reasons for signal integrity violations is the close placement of simultaneously switching elements. If a certain design layout solution requires that too many simultaneously switching elements are not put together in any of the bins, then we can assign the types for cells based on the time that particular cell switches and thus force simultaneously switching elements to be spread out. Such a layout solution also has the potential of reducing peak power dissipation hot spots. Finally, the printed circuit board (PCB) designs often require assigning portions of the netlist to multiple chips. This application as well as heterogeneous multi-chip configurations require the use of the multi-resource and multi-constraint partitioning algorithms described in this paper.

References

[1] C. Ababei and N. Selvakkumaran and K. Bazargan and G. Karypis. Multi-Objective Circuit Partitioning for Cutsize and Path-Based Delay Minimization. In *Proc. of ICCAD*, pages 181–185, 2002. Also available on WWW at URL http://www.cs.umn.edu/~karypis.

[2] C. J. Alpert, J. H. Huang, and A. B. Kahng. Multilevel circuit partitioning. In *Proc. of DAC*, 1997.

[3] V. Betz and J. Rose VPR: A New Packing, Placement and Routing Tool for FPGA Research. In *Proc. 7th Intl. Workshop on Field Programmable Logic and Applications*, 1997. Also available on WWW at URL http://www.eecg.toronto.edu/ vaughn/vpr/vpr.html

[4] A. Caldwell, A. Kahng, and I. Markov. Can recursive bisection alone produce routable placements? In *Proc. of DAC*, pages 477–482, 2000.

[5] A. Caldwell, A. Kahng, and I. Markov Implications of Area-Array I/O for Row-Based Placement Methodology in *Proc. of IEEE Symp. IC/Package Design Integration*, pages 93-98, 1998.

[6] J. Cong, H. P. Li, S. K. Lim, T. Shibuya, and D. Xu. Large scale circuit partitioning with loose/stable net removal and signal flow based clustering. In *Proc. of DAC*, pages 441–446, 1997.

[7] J. Cong and S.K.Lim. Edge separability based circuit clustering with application to circuit partitioning. In *Proc. ASP-DAC*, pages 429–434, 2000.

[8] J. Cong(Editor) and J. Shinnerl(Editor). Chapter3: Multilevel hypergraph partitioning. In *Multilevel Optimization in VLSICAD*, 2003.

[9] C. M. Fiduccia and R. M. Mattheyses. A linear time heuristic for improving network partitions. In *Proc. of DAC*, pages 175–181, 1982.

[10] R. Jayaraman Physical Design for FPGAs In *Proc. of ISPD*, pages 214-221, 2001.

[11] G. Karypis. Multilevel algorithms for multi-constraint hypergraph partitioning. Technical Report TR 99-034, Department of Computer Science, University of Minnesota, 1999. Also available on WWW at URL http://www.cs.umn.edu/~karypis.

[12] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Application in vlsi domain. *IEEE Transactions on VLSI Systems*, 20(1), 1999. A short version appears in the proceedings of DAC 1997.

[13] G. Karypis and V. Kumar. Multilevel *k*-way Hypergraph Partitioning. *Proceedings of the Design and Automation Conference*, 1999.

[14] G. Karypis, V. Kumar. A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, 20(1), 1999. Also available on WWW at URL http://www.cs.umn.edu/~karypis. A short version appears in Intl. Conf. on Parallel Processing 1995.

[15] G. Karypis and V. Kumar. hMeTiS 1.5: A hypergraph partitioning package. Technical report, Department of Computer Science, University of Minnesota, 1998. Available on the WWW at URL *http://www.cs.umn.edu/~metis*.

[16] G. Karypis and V. Kumar. Multilevel algorithms for multi-constraint graph partitioning. In *Proceedings of Supercomputing*, 1998. Also available on WWW at URL http://www.cs.umn.edu/~karypis.

[17] B. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. In *Bell Systems Technical Journal Vol 49-2*, pages 291-308, 1970.

[18] R. Kuznar and F. Brglez and B. Zajc Multi-way Netlist Partitioning into Heterogeneous FPGAs and Minimization of Total Device Cost and Interconnect. In *Proc. of DAC*, pages 238-244, 1994.

[19] P. Madden. FengShui Placement Tool http://vlsicad.cs.binghamton.edu/software.html

[20] I. Ouaiss and S. Govindarajan and V. Srinivasan and M. Kaul and R. Vemuri An Integrated Partitioning and Synthesis System for Dynamically Reconfigurable Multi-FPGA Architectures. In *Proc. of 5th Reconfigurable Architectures Workshop*, 1998.

[21] H. Liu, K. Zhu, and D. Wong. Circuit partitioning with complex resource constraints in FPGAs. In *Proc. of FPGA*, pages 77–84, 1998.

[22] N. Selvakkumaran and G. Karypis THETO - A Fast and High-Quality Partitioning Driven Global Placer In *University of Minnesota - Computer Science and Engineering Technical Reports*, 03-46, 11/25/2003. http://www.cs.umn.edu/tech_reports/index.cgi?selectedyear=2003&mode=printreport&report_id=03-046 Also available on WWW at URL http://www.cs.umn.edu/~karypis.

[23] N. Selvakkumaran and A. Ranjan and S. Raje and G. Karypis Multi-Resource Aware Partitioning Algorithms for FPGAs with Heterogeneous Resources In *Proc. DAC 2004*, pages 741-746, 2004.

[24] T. Taghavi and S. Ghiasi and A. Ranjan and S. Raje and M. Sarrafzadeh Innovate or Perish: FPGA Physical Design In *Proc. of ISPD*, pages 148-155, 2004.

[25] M. C. Walshaw and K. McManus. Multiphase mesh partitioning. *Applied Mathematical Modelling*, 25:123–140, 2000.

[26] M. Wang, A. Ranjan, and S.Raje. Multi-million gate FPGA physical design challenges. In *Proc. ICCAD 2003*, pages 891–898, 2003.

[27] M. Wang, X. Yang, and M. Sarrafzadeh. Dragon2000: Standard-cell placement tool for large industry circuits. In *Proc. of ICCAD*, pages 160–163, 2000.

[28] S. Wichlund and E. J. Aas. On Multilevel Circuit Partitioning. In *Proc. of ICCAD*, 1998.

[29] P. Zuchowski, C. Reynolds, R. Grupp, S. Davis, B. Cremen, and B. Troxel. A hybrid ASIC and FPGA architecture. In *Proc. of ICCAD*, pages 187–194, 2002.